AP® COMPUTER SCIENCE A
2016 GENERAL SCORING GUIDELINES

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)

w) Extraneous code that causes side-effect (e.g., writing to output, failure to compile)

x) Local variables used but none declared

y) Destruction of persistent data (e.g., changing value referenced by parameter)

z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. <>
- = instead of == and vice versa
- `length`/`size` confusion for array, String, List, or ArrayList; with or without ( )
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing ; where structure clearly conveys intent
- Missing { } where indentation clearly conveys intent
- Missing ( ) on parameter-less method or constructor invocations
- Missing ( ) around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context. For example,* "ArayList" *instead of* "ArrayList"*. As a counter example, note that if the code declares* "Bug bug;"*, then uses* "Bug.move()" *instead of* "bug.move()"*, the context does **not** allow for the reader to assume the object instead of the class.*

AP® COMPUTER SCIENCE A
2016 SCORING GUIDELINES

Question 3: Crossword

Part (a) `toBeLabeled` **3 points**

Intent: *Return a `boolean` value indicating whether a crossword grid square should be labeled with a positive number*

+1 Checks `blackSquares[r][c]`

+1 Checks for black square/border above and black square/border to the left (*no bounds errors*)

+1 Returns `true` if square should be labeled with positive number; returns `false` otherwise

Part (b) `Crossword` constructor **6 points**

Intent: *Initialize each square in a crossword puzzle grid to have a color `(boolean)` and an integer label*

+1 `puzzle = new Square[blackSquares.length][blackSquares[0].length];` (*or equivalent*)

+1 Accesses all locations in `puzzle` (*no bounds errors*)

+1 Calls `toBeLabeled` with appropriate parameters

+1 Creates and assigns new `Square` to location in `puzzle`

+1 Numbers identified squares consecutively, in row-major order, starting at 1

+1 On exit: All squares in `puzzle` have correct color and number (*minor errors covered in previous points ok*)

Question-Specific Penalties

-2 (p) Consistently uses incorrect name instead of `puzzle`

-1 (q) Uses *array*`[].length` instead of *array*`[`*num*`].length`

AP® COMPUTER SCIENCE A
2016 CANONICAL SOLUTIONS

Question 3: Crossword

Part (a):

```
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
{
      return (!(blackSquares[r][c]) &&
              (r == 0 || c == 0 || blackSquares[r - 1][c] ||
               blackSquares[r][c - 1]));
}
```

Part (b):

```
public Crossword(boolean[][] blackSquares)
{
      puzzle = new Square[blackSquares.length][blackSquares[0].length];
      int num = 1;

      for (int r = 0; r < blackSquares.length; r++)
      {
            for (int c = 0; c < blackSquares[0].length; c++)
            {
                  if (blackSquares[r][c])
                  {
                        puzzle[r][c] = new Square(true, 0);
                  }
                  else
                  {
                        if (toBeLabeled(r, c, blackSquares))
                        {
                              puzzle[r][c] = new Square(false, num);
                              num++;
                        }
                        else
                        {
                              puzzle[r][c] = new Square(false, 0);
                        }
                  }
            }
      }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

Complete method `toBeLabeled` below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
{
 if(blackSquares[r][c] == true) return false;
 else if (r == 0) return true;
 else if (c == 0) return true;
 else if (blackSquares[r-1][c] == true) return true;
 else if (blackSquares[r][c-1] == true) return true;
 else return false;
}
```

Part (b) begins on page 18.

Assume that `toBeLabeled` works as specified, regardless of what you wrote in part (a). You must use `toBeLabeled` appropriately to receive full credit.

Complete the `Crossword` constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    - The crossword puzzle grid has the same dimensions as blackSquares.
 *    - The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    - The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
{
puzzle = new Square[blackSquares.length][blackSquares[0].length];
int rows = puzzle.length;
int cols = puzzle[0].length;
int label = 1;
for(int r = 0; r < rows; r++)
{
 for(int c=0; c < cols; c++)
 {
  if(blackSquares[r][c] == true)
     puzzle[r][c] =      Square(true, 0);
  else if(toBeLabeled(r, c, blackSquares) == true)
  {
   puzzle[r][c] =      Square(false, label);
   label ++;
  }
  else puzzle[r][c] =      Square(false, 0);
 }
```

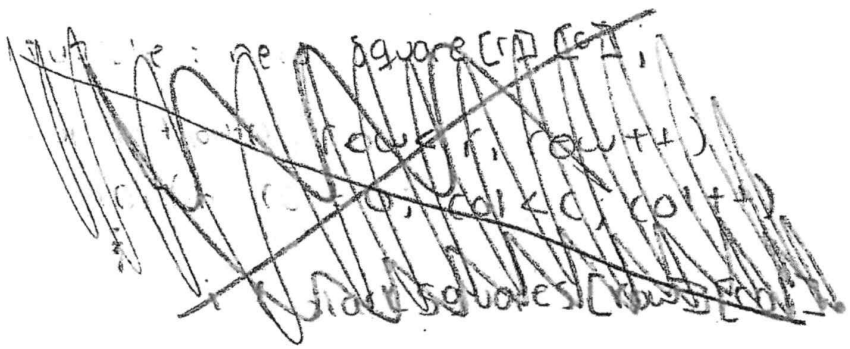

GO ON TO THE NEXT PAGE.

3Ba

Complete method `toBeLabeled` below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
```

{



```
if (r==0 || c==0)
    return true;

if( blackSquares[r-1][c] == true ||
    blackSquares[r][c-1] == true)
    return true;

return false;
```

}

Part (b) begins on page 18.

Assume that `toBeLabeled` works as specified, regardless of what you wrote in part (a). You must use `toBeLabeled` appropriately to receive full credit.

Complete the `Crossword` constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    - The crossword puzzle grid has the same dimensions as blackSquares.
 *    - The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    - The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
{
    int num = 1;
    puzzle = new Square[blackSquares.length]
                       [blackSquares[0].length];

    for(int r=0; r<blackSquares.length; r++)
        for(int c=0; c<blackSquares[0].length; c++)
        {
            if(blackSquares[r][c]==true)
                puzzle[r][c]= new Square(true, 0);
            else
                puzzle[r][c]= new Square(true, num);
                num++;
            }
        }
}
```

3Ca

Complete method `toBeLabeled` below.

```
/** Returns true if the square at row r, column c should be labeled with a positive number;
 *          false otherwise.
 *  The square at row r, column c is black if and only if blackSquares[r][c] is true.
 *  Precondition: r and c are valid indexes in blackSquares.
 */
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
{
  if( blackSquares[r][c])
      return false;
  else if((c=0 ||blackSquare[r][c-1]) && (r=0 ||blackSquare[r-1][c]))
      return true;
  else return false;
}
```

Part (b) begins on page 18.

Assume that `toBeLabeled` works as specified, regardless of what you wrote in part (a). You must use `toBeLabeled` appropriately to receive full credit.

Complete the `Crossword` constructor below.

```
/** Constructs a crossword puzzle grid.
 *  Precondition: There is at least one row in blackSquares.
 *  Postcondition:
 *    - The crossword puzzle grid has the same dimensions as blackSquares.
 *    - The Square object at row r, column c in the crossword puzzle grid is black
 *      if and only if blackSquares[r][c] is true.
 *    - The squares in the puzzle are labeled according to the crossword labeling rule.
 */
public Crossword(boolean[][] blackSquares)
{
  Square[][] puzzle = new Square[blackSquares.length][ ];
                                                  blackSquares[c].length()

  for(int r = 0; r < blackSquares.length; r++)
  { for(int c=0; c < blackSquares[0].length; c++)
      if ( toBeLabeled(r, c, blackSquares[r][c]))
          puzzle[r][c] = new Square[!isBlack, 1];

      else puzzle[r][c] = new Square[isBlack, 0];
  }
```

AP® COMPUTER SCIENCE A
2016 SCORING COMMENTARY

Question 3

Overview

This question used a two-dimensional (2-D) array of objects to represent a crossword puzzle.

In part (a) students were asked to write a `boolean` method that determines whether or not a specific square in the puzzle should be numbered based on specified labeling rules. Students needed to demonstrate an understanding of `boolean` data structures, writing and evaluating `boolean` expressions, and returning correct `boolean` values. Students were also required to demonstrate an understanding of bounds checking and indexing in a 2-D array. This logic needed to be implemented utilizing given method parameters.

In part (b) students were asked to write a constructor that initializes the instance variable and explicitly calls the method defined in part (a) to build a puzzle, square by square. Students were required to demonstrate an understanding of object instantiation by initializing the class instance variable and a `Square` object using appropriate parameter values. Students were required to demonstrate an understanding of 2-D array processing by traversing a 2-D array in row-major order, accessing each position of the array without going out of bounds. Students were also required to identify squares that needed to be consecutively numbered by calling the previously defined `toBeLabeled` method using appropriate parameters.

Sample: 3A
Score: 8

In part (a) the student checks the square at row `r`, column `c` and correctly implements the crossword labeling rules by checking for a white square in the first row or first column, or for a white square with a black square to its left or above. No checks cause a bounds error. The logic correctly returns `true` if the square should be numbered and `false` otherwise. Part (a) earned 3 points.

In part (b) the student correctly instantiates the instance variable `puzzle` with the same dimensions as the parameter `blackSquares`. All locations in `puzzle` are accessed without bounds errors and assigned a `Square` object, but the assignment is missing the `new` operator, so the "assign new `Square`" point was not earned. Labeled squares, identified by a correct call to the `toBeLabeled` method, are numbered consecutively in row-major order starting at 1. Squares that are not labeled are numbered with 0. Part (b) earned 5 points.

Sample: 3B
Score: 5

In part (a) the student fails to check the square at row `r`, column `c`, so the first rubric point was not earned. The square above and the square to the left are checked with no bounds error, so the second rubric point was earned. Because the logic does not include checking the color of the square to be labeled, the third rubric point was not earned. Part (a) earned 1 point.

In part (b) the student correctly instantiates the instance variable `puzzle` with the same dimensions as the parameter `blackSquares`. All locations in `puzzle` are accessed without bounds errors and assigned a correctly constructed `Square` object. The student fails to call the `toBeLabeled` method, so the "calls `toBeLabelled`" point was not earned. Squares are numbered consecutively in row-major order starting at 1. Squares that are not labeled are numbered with 0. Because the squares to be labeled are not correctly identified (and because all created squares are black), the final point was not earned. Part (b) earned 4 points.

AP® COMPUTER SCIENCE A
2016 SCORING COMMENTARY

Question 3 (continued)

Sample: 3C
Score: 3

In part (a) the student checks the square at row `r`, column `c` and correctly implements the crossword labeling rules by checking for a white square in the first row or first column, or for a white square with a black square to its left or above. No checks cause a bounds error. The logic used for the return value is incorrect. In order to be labeled, inner white squares will require a black square both above and to the left, but because only one adjacent black square is required, the student did not earn the third rubric point. Part (a) earned 2 points.

In part (b) the student incorrectly includes `Square[][]` to declare a local object rather than instantiating the instance variable, `puzzle`, so the "initialize `puzzle`" point was not earned. All locations in `puzzle` are accessed without bounds errors and assigned a `Square` object, but the constructor call to `Square` has an unknown variable as its first parameter, so the "assign new `Square`" point was not earned. Labeled squares, identified by a call to the `toBeLabeled` method, are numbered with either 1 or 0, so the "number identified squares" point was not earned. The call to the `toBeLabeled` method uses a `boolean` value as the third parameter rather than a 2-D `boolean` array value, so the "calls `toBeLabelled`" point was not earned. Unlabeled white squares are not numbered with 0, so the final point was not earned. Part (b) earned 1 point.